



AuthGuard: A Secure Authentication Framework Against Query Manipulation Attacks

K. Vijay Kumar¹, Jatavath Chandu¹, Pasala Gnana Abhishek¹, Vatti Karthik Reddy¹, J. Hemanth Krishna Nivas¹

¹Department of Computer Science and Engineering, ¹Sree Dattha Institute of Engineering and Science, Nagarjuna Sagar Road, Sheriguda, Ibrahimpatnam, Rangareddy Dist, 501510, Telangana, India.

ABSTRACT

The rapid growth of web-based applications has significantly increased the need for secure authentication mechanisms to protect user data from cyber threats such as Structured Query Language injection (SQLi) attacks. This research focuses on the design and development of a Django-based web application that demonstrates both vulnerable and secure login systems to highlight the importance of secure coding practices. Many existing authentication systems construct SQL queries dynamically without proper input validation, making them susceptible to injection attacks that can lead to unauthorized access and data breaches. Traditional approaches often prioritize functionality over security, relying on unsafe query concatenation and weak validation techniques, which fail to effectively detect malicious inputs. To address these challenges, the proposed system implements a dual-module architecture consisting of a vulnerable login module and a secured login module. The vulnerable module illustrates how improper input handling can be exploited, while the secured module incorporates input validation and basic attack detection mechanisms to identify suspicious patterns in user credentials. The system is developed using the Django framework with a MySQL database for efficient user management and authentication. The primary objective of this research is to provide a practical understanding of web application vulnerabilities and their mitigation techniques. By enabling comparative analysis between insecure and secure implementations, the system promotes awareness of secure authentication practices. The proposed solution enhances data integrity, improves system reliability, and serves as a foundational model for developing secure web applications resistant to common cyberattacks.

Keywords: Structured Query Language injection (SQLi) Attack, Secure Authentication, Input Sanitization, Web Security, Vulnerability Analysis, Cyber Threats, MySQL Database, Secure Web Development

1. INTRODUCTION

SQLi vulnerability detection has become a research hot spot in information security. Currently, the mainstream approach is to use static analysis techniques to detect SQLi vulnerabilities. Static analysis technology refers to analysing web application code without executing it, mainly relying on syntax and semantic features, such as abstract syntax tree (AST), control flow graph (CFG), and call flow graph, matching according to the characteristics of vulnerabilities to discover them. After a long development, static analysis techniques have become increasingly mature and have been used to automate the detection of SQLi vulnerabilities with certain results. However, existing static analysis techniques cannot accurately detect SQLi vulnerabilities in applications that use object-oriented database extension (OODBE). A WAV is defined as “a flaw in the application that stems from coding defects and causes severe damage to the application upon exploitation”. To identify and mitigate vulnerabilities that may be exploited by attackers, a penetration testing method or ethical hacking is used. The Open Web Application Security Project (OWASP) provides the standard for such penetration testing methodology to test web applications and could be used to evaluate the effectiveness of web vulnerability scanners. Web Application Vulnerability Scanners (WAVS) are tools used by penetration testers. It is used to conduct web application evaluations with the primary goal of identifying and mitigating potential vulnerabilities to prevent security breaches. Security measures should be incorporated throughout the



development life cycle rather than being added and tested only at the final stages of the development life cycle. The challenge here is that when using different WAVS, they will not offer the same vulnerability results for the same target. These differences arise from the different levels of precision, speed, and coverage in terms of finding various vulnerabilities, for example, with the respect to XSS and SQLi attacks. These differing results mean that penetration testers need to utilise multiple WAVS and thus, the accuracy and detection coverage of the penetration testing report depend on the testers' knowledge and experience of the most effective WAVS in particular situations. This manual approach to Web Application Penetration Testing (WAPT) can be time-consuming, costly, and subject to human error.

2. Related Work

The field of penetration testing has evolved significantly from manual security assessments to automated and intelligent frameworks incorporating Artificial Intelligence (AI), Machine Learning (ML), and advanced analytics. Early approaches primarily relied on human expertise and rule-based tools, which were time-consuming and limited in scalability. Recent research focuses on improving efficiency, accuracy, and adaptability of vulnerability detection systems through automation and intelligent techniques.

2.1 Automated Penetration Testing

Automation has become a key focus in reducing dependency on human expertise. Shahid et al. [2] evaluated automated web application penetration testing tools, demonstrating their ability to reduce time and cost while highlighting limitations such as incomplete scanning and inaccurate results. Moreira et al. [7] developed a low-cost automated vulnerability detection system requiring minimal user intervention, validated on real-world applications. Similarly, Seara et al. [8] introduced a user-friendly vulnerability scanner designed for both experts and non-experts, addressing the shortage of cybersecurity professionals.

2.2 AI and Machine Learning in Security

The integration of AI and ML has significantly improved the detection of complex vulnerabilities. Moreno et al. [4] demonstrated that ML techniques can effectively identify hidden malicious patterns, although real-world environments remain complex. Alaoui et al. [10] conducted a systematic review showing that Deep Learning (DL) models outperform traditional approaches in detecting unknown attacks. Furthermore, Chowdhary et al. [15] proposed a GAN-based framework capable of generating realistic attack payloads, improving automation and scalability in penetration testing.

2.3 Intelligent and Reinforcement Learning-Based Approaches

Recent studies have explored intelligent and adaptive penetration testing strategies. Li et al. [12] proposed a reinforcement learning-based model using a Network Model for Penetration Testing (NMPT) combined with Markov Decision Processes to simulate adaptive attack behaviors. Pratama et al. [13] introduced CIPHER, a domain-specific large language model trained to assist ethical hackers by providing expert-level guidance in penetration testing tasks. Xiao et al. [1] further advanced this direction with SatGuard, integrating LLMs to enable semi-automated vulnerability analysis and exploitation.

2.4 Web Application Security and Vulnerability Analysis

Web-based systems remain a major target for cyberattacks. Nam et al. [6] demonstrated how vulnerabilities in non-encrypted APIs can lead to unauthorized data access and information leakage. Altulaihan et al. [3] provided a comprehensive overview of web penetration testing techniques and associated vulnerabilities. Althunayyan et al. [11] evaluated modern black-box scanners against injection attacks such as SQLi, NoSQL, and SSTI, identifying key limitations in detection accuracy.

2.5 Vulnerability Management and Security Frameworks

Effective vulnerability management is critical for maintaining system security. Sarker et al. [5] emphasized the importance of regular penetration testing to identify system weaknesses and ensure



security compliance. Bennouk et al. [9] highlighted the role of proactive vulnerability management systems in mitigating cyber threats, while Sotiropoulos et al. [14] proposed a framework for prioritizing vulnerabilities based on impact, remediation cost, and available resources, particularly in IoT environments.

2.6 Research Gap

Although significant progress has been made in automated and AI-driven penetration testing, several limitations still exist. Most existing solutions either rely heavily on predefined rules, lack adaptability to evolving threats, or require significant computational resources and expert intervention. Additionally, many tools fail to provide accurate and complete vulnerability coverage in real-world environments. There is a need for a lightweight, intelligent, and scalable penetration testing framework that integrates automation with adaptive learning capabilities while minimizing human dependency. The proposed system aims to address these challenges by developing an efficient and intelligent penetration testing solution with improved accuracy and scalability.

3. PROPOSED METHODOLOGY

The proposed research focuses on understanding how web-based authentication mechanisms behave under normal usage as well as malicious conditions. It examines how user credentials are collected, processed, and validated against a backend database, highlighting the interaction between the application layer and the database layer. Special attention is given to how improper handling of user inputs can expose systems to serious security threats. By observing both normal user behaviour and crafted malicious inputs, the study emphasizes the importance of secure coding practices in real-world web applications as shown in fig 1.

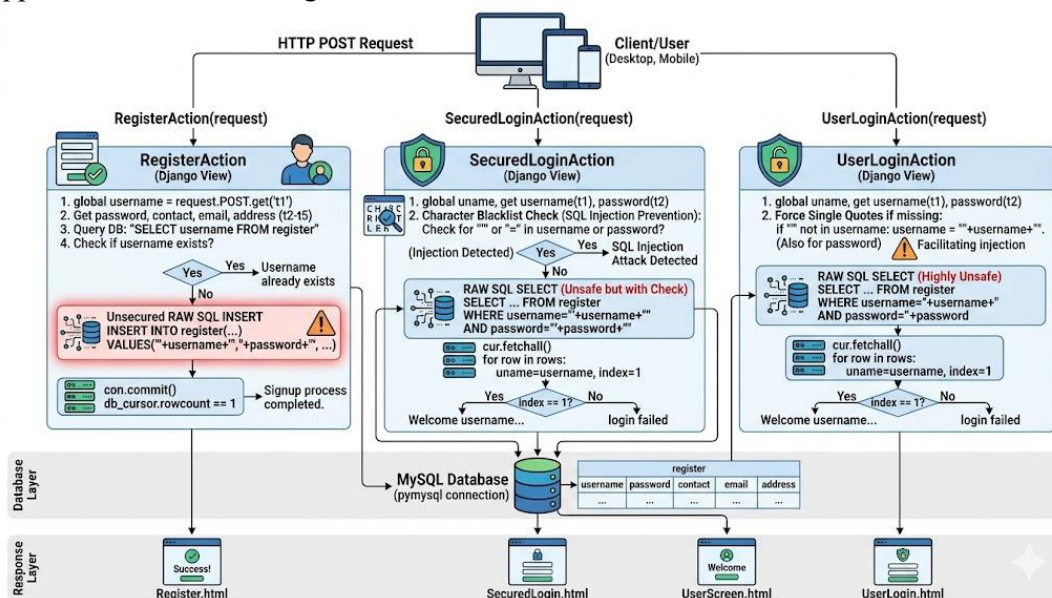


Fig. 1: Proposed system architecture of real-time SQLi defense.

The study further analyzes different authentication approaches to compare vulnerable and protected access mechanisms. One approach demonstrates how direct query construction using user input can allow unauthorized access through manipulation techniques, while another approach introduces basic validation checks to identify suspicious patterns. This comparative analysis helps in understanding how attackers exploit weak input handling and how even simple defensive checks can reduce risk, though not eliminate it entirely. The behavior of the system under attack scenarios provides clear insights into common weaknesses present in beginner-level authentication implementations.

Secured Login Workflow

The secured login workflow operates by first passing user input through validation filters that remove or block suspicious characters. Only after inputs are verified, the SQL query is executed in a controlled



manner using clean parameters within the query. Because the detection layer cancels harmful input upfront, the final SQL statement cannot be manipulated or injected with malicious patterns. As a result, the database returns accurate and safe results allowing login only when credentials match exactly and redirects the user accordingly while maintaining protection despite using string concatenation as shown in fig 2.

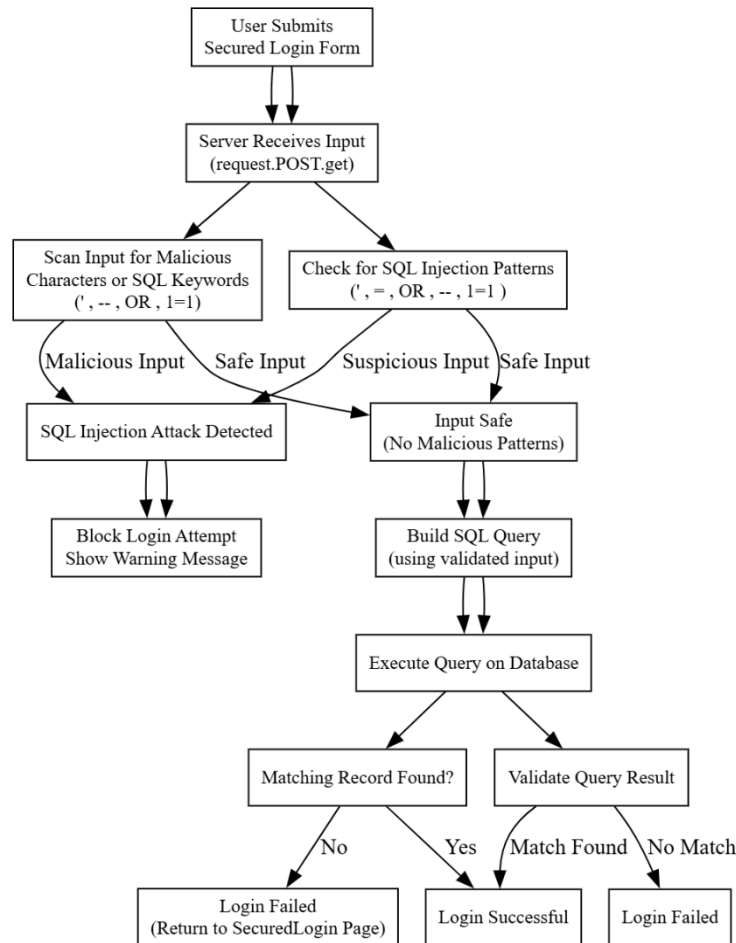


Fig. 2: secured login workflow

User Submits Secured Login Form: The secured login workflow begins when the user enters their username and password in the secured login interface and submits the form. These credentials are sent to the server via a POST request. Unlike the vulnerable login, this workflow is specifically designed to detect suspicious input patterns before interacting with the database.

Server Receives User Credentials: Once the form data reaches the Django backend, the server extracts the username and password using `request.POST.get()`. These values are temporarily stored in variables for validation. At this stage, the system prepares to analyse the input to detect any signs of SQLi attempts before further processing.

Input Validation and Malicious Pattern Detection: The server checks the username and password for common SQLi indicators such as single quotes (`'`), double quotes, equal signs (`=`), or logical operators. If any of these characters or patterns are detected, the system immediately identifies the request as a potential SQLi attack. This preventive mechanism ensures that malicious payloads never reach the query-building phase.

Block Attack and Display Warning Message: If suspicious characters are found, the server blocks the login process right away. Instead of sending a query to the database, the system generates a security alert message such as “SQLi Attack Detected.” This response helps protect the backend by preventing harmful code from being executed and keeps the application safe from unauthorized access attempts.



Proceed with Clean Input: If no malicious patterns are detected, the workflow continues with the validated input. The server now considers the username and password safe enough to use in the SQL query. This step forms the basis of the secured login process, ensuring that only clean, human-intended input is forwarded for authentication.

Execute SQL Query to Validate Credentials: The backend constructs and executes a SQL SELECT query to check whether the provided credentials match any records in the database. Although string concatenation is still used, the prior input validation greatly reduces the risk of injection attacks. The database returns user details only if the credentials are correct.

Check Query Result and Determine Login Status: Once the query is executed, the server inspects the returned rows. If at least one matching record is found, the login is considered valid, and the user is authenticated successfully. If no records are returned, it indicates incorrect credentials, and the login attempt is denied.

Display Login Success or Failure Message: Depending on the query results, the system generates an appropriate response. A successful authentication leads the user to the welcome screen, confirming entry into the system. If authentication fails, the system reloads the secured login page with a message such as “Login Failed,” prompting the user to retry.

4. IMPLEMENTATION DESCRIPTION

This chapter describes how the core Python code of the Dynamic Exploit-and-Defense Framework has been implemented using Django. The system consists of two main modules a vulnerable login flow to demonstrate SQLi and a secured login flow to detect and block attacks along with user registration, simple page-rendering views, and research configuration.

Environment & Dependencies

- **Python version:** 3.7.6
- **Django:** for rapid web-app scaffolding and template rendering.
- **PyMySQL:** to establish direct connections to the MySQL database named vulner.
- **Templates:** HTML pages (index.html, Register.html, UserLogin.html, SecuredLogin.html, UserScreen.html) stored in the app’s templates directory.

RegisterAction(request)

- **Input:** Expects POST data fields t1 through t5 for username, password, contact, email, and address.
- **Logic:** Opens a MySQL connection with PyMySQL; fetches all existing usernames to check for duplicates.
- **Insertion:** If no duplicate is found, builds an INSERT query by concatenating user inputs (demonstrating unsafe practice) and commits the record.
- **Output:** Returns a context variable data that displays either “Username already exists” or “Signup process completed” to Register.html.

Secured Login Action(request)

- **Input:** POST fields t1 (username) and t2 (password).
- **SQL-Injection Check:** Scans inputs for suspicious characters (' or =). If found, immediately blocks the attempt and renders SecuredLogin.html with “SQLi Attack Detected.”
- **Safe Query:** Otherwise, performs a string concatenated SELECT against register. If a match is found, renders UserScreen.html with a welcome message; if not, returns “login failed.”

User Login Action(request)

- **Input:** POST fields t1, t2.
- **Intentional Vulnerability:** Ensures inputs are wrapped in single quotes, then executes a concatenated SELECT query—fully vulnerable to injection (e.g., OR 1=1).
- **Outcome:** Successful match logs the user in; failure returns “login failed” on UserLogin.html.

Render-only Views



- Register, UserLogin, index, SecuredLogin: Each handles GET requests and simply renders the corresponding template with no back-end logic.

Configuration (settings.py)

- SECRET_KEY and DEBUG are set for development.
- INSTALLED_APPS includes Django's core modules plus VulnerApp.
- MIDDLEWARE enables security, session, CSRF, and authentication layers.
- DATABASES: Although Django's default is configured for SQLite3, this research bypasses the ORM in views and uses PyMySQL to connect to MySQL—highlighting an architectural inconsistency for demonstration purposes.
- STATIC_URL and internationalization settings remain at their defaults.

Data Flow & Security Considerations

- **Data Flow:** Users interact via the browser, submitting forms that trigger Django views. Views construct SQL queries and communicate with MySQL. Responses are rendered back into HTML templates.
- **Vulnerability Demonstration:** The UserLoginAction view illustrates how direct query concatenation can be exploited.
- **Basic Defense:** The SecuredLoginAction view shows a naive input-validation approach to block obvious injection patterns.
- **Next Steps:** In a production version, all raw SQL should be replaced with Django's ORM or parameterized queries, and the hard-coded credentials and DEBUG=True setting must be secured via environment variables.

5. CONCLUSION

This work demonstrates the critical importance of secure authentication mechanisms in modern web applications by presenting a comparative analysis between a vulnerable login system and a secured module protected against SQLi attacks. The study clearly illustrates how improper input handling and unsafe query construction can be exploited by attackers to bypass authentication and manipulate database operations. The implementation highlights the limitations of traditional approaches that rely on direct query concatenation and insufficient validation techniques. By utilizing the Django framework with PyMySQL, the system provides a practical and realistic environment to understand both the occurrence of security vulnerabilities and their mitigation. The secured login module incorporates input validation and basic attack detection mechanisms, effectively preventing unauthorized access by filtering suspicious patterns. The research reinforces the necessity of adopting secure coding practices such as proper input validation, structured database interactions, and avoidance of dynamic SQL query construction. It contributes to improving awareness of web security challenges and provides a foundational approach for developing robust and secure authentication systems resistant to common cyber threats.

REFERENCE

- [1] Xiao J, Wang B, Dong R, Zhao Z, Zhao B. SatGuard: Satellite Networks Penetration Testing and Vulnerability Risk Assessment Methods. Aerospace. 2025; 12(5):431. <https://doi.org/10.3390/aerospace12050431>
- [2] Shahid J, Hameed MK, Javed IT, Qureshi KN, Ali M, Crespi N. A Comparative Study of Web Application Security Parameters: Current Trends and Future Directions. Applied Sciences. 2022; 12(8):4077. <https://doi.org/10.3390/app12084077>
- [3] Altulaihan EA, Alismail A, Frikha M. A Survey on Web Application Penetration Testing. Electronics. 2023; 12(5):1229. <https://doi.org/10.3390/electronics12051229>
- [4] Moreno AC, Hernandez-Suarez A, Sanchez-Perez G, Toscano-Medina LK, Perez-Meana H, Portillo-Portillo J, Olivares-Mercado J, García Villalba LJ. Analysis of Autonomous Penetration



- Testing Through Reinforcement Learning and Recommender Systems. *Sensors*. 2025; 25(1):211. <https://doi.org/10.3390/s25010211>
- [5] Sarker KU, Yunus F, Deraman A. Penetration Taxonomy: A Systematic Review on the Penetration Process, Framework, Standards, Tools, and Scoring Methods. *Sustainability*. 2023; 15(13):10471. <https://doi.org/10.3390/su151310471>
- [6] Nam Y, Choi S. Analysis of Vulnerabilities in College Web-Based System. *Electronics*. 2024; 13(12):2261. <https://doi.org/10.3390/electronics13122261>
- [7] Moreira D, Seara JP, Pavia JP, Serrão C. Intelligent Platform for Automating Vulnerability Detection in Web Applications. *Electronics*. 2025; 14(1):79. <https://doi.org/10.3390/electronics14010079>
- [8] Seara JP, Serrão C. Automation of System Security Vulnerabilities Detection Using Open-Source Software. *Electronics*. 2024; 13(5):873. <https://doi.org/10.3390/electronics13050873>
- [9] Bennouk K, Ait Aali N, El Bouzekri El Idrissi Y, Sebai B, Faroukhi AZ, Mahouachi D. A Comprehensive Review and Assessment of Cybersecurity Vulnerability Detection Methodologies. *Journal of Cybersecurity and Privacy*. 2024; 4(4):853-908. <https://doi.org/10.3390/jcp4040040>
- [10] Alaoui RL, Nfaoui EH. Deep Learning for Vulnerability and Attack Detection on Web Applications: A Systematic Literature Review. *Future Internet*. 2022; 14(4):118. <https://doi.org/10.3390/fi14040118>
- [11] Althunayyan M, Saxena N, Li S, Gope P. Evaluation of Black-Box Web Application Security Scanners in Detecting Injection Vulnerabilities. *Electronics*. 2022; 11(13):2049. <https://doi.org/10.3390/electronics11132049>
- [12] Li Y, Wang Y, Xiong X, Zhang J, Yao Q. An Intelligent Penetration Test Simulation Environment Construction Method Incorporating Social Engineering Factors. *Applied Sciences*. 2022; 12(12):6186. <https://doi.org/10.3390/app12126186>
- [13] Pratama D, Suryanto N, Adiputra AA, Le T-T-H, Kadiptya AY, Iqbal M, Kim H. CIPHER: Cybersecurity Intelligent Penetration-Testing Helper for Ethical Researcher. *Sensors*. 2024; 24(21):6878. <https://doi.org/10.3390/s24216878>
- [14] Sotiropoulos P, Mathas C-M, Vassilakis C, Kolokotronis N. A Software Vulnerability Management Framework for the Minimization of System Attack Surface and Risk. *Electronics*. 2023; 12(10):2278. <https://doi.org/10.3390/electronics12102278>
- [15] Chowdhary A, Jha K, Zhao M. Generative Adversarial Network (GAN)-Based Autonomous Penetration Testing for Web Applications. *Sensors*. 2023; 23(18):8014. <https://doi.org/10.3390/s23188014>